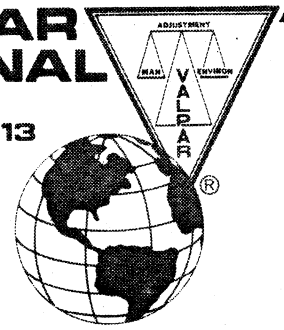
VALPAR
INTERNATIONAL
3801 E. 34TH STREET
TUCSON, ARIZONA 85713
602-790-7141



valFORTH™
SOFTWARE SYSTEM
for ATARI*

PLAYER-MISSILE GRAPHICS

*Atari is a trademark of Atari, Inc., a division of Warner Communications.

valFORTH™
SOFTWARE SYSTEM

PLAYER-MISSILE GRAPHICS

Stephen Maguire

valFORTH T.M.

PLAYER-MISSILE GRAPHICS

Version 1.0
April 1982

The following is a description of commands used in creating seemingly difficult video displays using players and missiles. Used alone or in combination with the other available systems by Valpar International, it is possible to obtain graphic displays which compare with those of the best arcade games. The use of players and missiles (also called "player/missiles") allows the beginner to create high quality moving video displays.

VALPAR INTERNATIONAL

Disclaimer of Warranty
on Computer Programs

All Valpar International computer programs are distributed on an "as is" basis without warranty of any kind. The total risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

Valpar International shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by Valpar International. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

Defective media (diskettes) will be replaced if diskette(s) is returned to Valpar International within 30 days of date of sale to user.

Defective media (diskettes) which is returned after the 30 day sale date will be replaced upon the receipt by Valpar of a $12.00 Replacement Fee.

PLAYER/MISSILE GRAPHICS PACKAGE

As knowledge of the internal workings of player/missile graphics is not necessary to use this valFORTH package effectively, the internal workings are not explained in this manual. However, for the serious programmer trying to optimize his/her program in every way, an understanding of these internal workings could at times improve code efficiency and/or speed of execution. For a complete explanation of player/missile graphics at the nut-and-bolt level, see the series of articles by Dave and Sandy Small in Creative Computing.

STROLLING THROUGH PLAYER/MISSILE GRAPHICS

One of the biggest differences between the Atari graphic capabilities and those of most other computers is the Atari's ability to use players and missiles. This discussion will not explain the internal workings of player/missile graphics on the Atari; rather, it will explain how to use the basic commands in this valFORTH package. Before we proceed, please load the player/missile graphic routines from the Player/Missile disk. The directory on screen 170 will show what screen to load. Also, if you have the valFORTH Editor/Utilities package, load in the high speed STICK command found in the Miscellaneous Utilities; otherwise, load in the slower version on your Player/Missile disk. (Check the directory for its location).

To start with, let's get a simple player up on the screen to experiment with. First we must initialize the player/missile graphic system and design the player's image. This is simple:

```
1  PMINIT                    ( Initialize for single
                               resolution players )

2  BASE  !                   ( Change to binary for ease )

LABEL CROSS                  ( Give the player image a name )
00011000  C,
00011000  C,
00011000  C,
11111111  C,                 ( A large plus sign )
11111111  C,
00011000  C,
00011000  C,
00011000  C,

DECIMAL                      ( Now back into base 10 )

PMCLR                        ( Clear player/missile memory )

ON PLAYERS                   ( Turn on the players )

CROSS  8  180  50  0  BLDPLY      ( Build a player )
```

You should now see the cross in the upper right-hand corner of the video screen. Now let's take a look at this and see how it works.

First, players are initialized using the PMINIT command. Players can be in either a single or double resolution mode (double res players are twice as tall). "1 PMINIT" is used for single res players. If we had wanted double res players, we would have used "2 PMINIT".

Next, the player image is created. Since it is much easier to make player images as 1's and 0's, we use binary (base two) number entry. Before we design the image, it must be given a name. The LABEL command does this nicely for us.

This image is named CROSS. All that need be done now is to draw the picture. Notice how easy it is to see the image when using base two. Of course, we could have stayed in base 10 and still designed the image, but this is usually more difficult. The word C, after each number simply tells FORTH to store that number in the dictionary. Once the picture is designed, we return to decimal for ease.

Both the PMCLR and ON PLAYERS commands are fairly self-descriptive: PMCLR erases all players and missiles so that no random trash appears when the PLAYERS are turned ON. Next, the BLDPLY (build player) command takes the image named CROSS which is 8 bytes tall and assigns it to player 0 at horizontal location 180 and vertical location 50 on the display. Of course, we could have built player 1, 2, or 3 instead.

The cross should be black. Suppose we wanted a blue or green cross instead. This can be done using the PMCOL (player/missile color) command. Try this:

```
0  9  8  PMCOL              ( player  hue  lum  PMCOL )
```

The cross should now appear blue. This command assigns a BLUE (9) hue with a luminance of 8 to player 0. If the color commands are loaded from the valFORTH disk,

```
0  BLUE  8  PMCOL
```

could have been used with the same results. Try changing the color of the player to GREEN (12) or PINK (4). Note that the default colors for players 2 and 3 make them invisible: Their colors should be set immediately upon being built.

Now that we have a player on the screen, let's move it around. We use the PLYMV (player move) command for this. PLYMV needs to know which player to move (there could be as many as five), how far to move it in the horizontal direction, and how far to move it in the vertical direction. Try this:

```
1  1  0  PLYMV            ( horz  vert  player  PLYMV )
```

This moves player 0 down 1 line and right one horizontal position, thus giving the effect of a diagonal move towards the lower right-hand corner. Try these as well:

```
 1   0  0  PLYMV          ( move right one position )
-5   0  0  PLYMV          ( move left five positions )
 0  20  0  PLYMV          ( move down 20 lines )
 0 -15  0  PLYMV          ( move up 15 lines )
-5   2  0  PLYMV          ( move left five, and down two )
```

That's all there is to moving a player. Positive horizontal offsets move the player right, and negative values move the player left. Likewise, positive vertical offsets move the player down while negative ones move the player up. The following program can be typed in and you will have a joystick controlled player:

```
: JOY
  BEGIN
    0 STICK                ( STICK leaves two offsets )
    0 PLYMV                ( for PLYMV to use. )
    ?TERMINAL
  UNTIL ;

JOY <ret>
```

Move the player with stick 0, the left-most stick port. Press any console button to exit the program.

Currently, if the player is moved off any edge, it "wraps" to the opposite side. In other words, we have an "unbound" player. This is rarely desirable. Normally, we want to restrict player movement to certain boundaries. The PLYMV command has a built in boundary check routine specifically for this reason. Right now, new boundaries are set so wrapping occurs. Let's set some boundaries:

```
60  150  50  200  0  PLYBND
```

This sets the boundaries of player zero to 75 on the left, 150 on the right, 50 on the top, and 200 on the bottom. Type JOY again to verify that you can no longer move freely about the display. Try different boundary settings and experiment to get the feel of the command. Boundary checking can be disabled for any or all of the edges. Setting the left or upper boundary to 0 will disable the check on that edge, likewise, 255 in either the right or lower boundary will do the same.

Let's build another player in the lower right-hand corner of the screen. This time, instead of designing the player ourself, let's borrow the image from the standard Atari character set stored in ROM. The image of the digit zero starts at address 57472. The other numbers follow zero. Try this:

```
57472  16  160  150  1  BLDPLY
```

You should now see the numbers 0 and 1 on your screen. This command builds player 1 with the image at address 57472 that is 16 bytes tall and puts it at horizontal position 160 and vertical position 150. Give this player a color if you want.

Until now, we have been using normal size players. It is possible to make the two players on the display different widths using the PLYWID command. PLYWID expects a width specification of 0 or 2 (normal), 1 (double), or 3 (quadruple). Its command form is:

```
width   player   PLYWID
```

Thus,

```
3  1  PLYWID
```

should make player one four times its original size. The same can be done with player zero:

```
3  0  PLYWID
```

Type JOY again and notice that the width has no effect on movement whatsoever. Also notice that player one is unaffected by movement of player zero.

Now that we have two players on the screen, let's interface both of them to the joystick. Type in the following program:

```
: JOY2
  BEGIN
    0  STICK                ( Record stick movement )
    2DUP                    ( Make a copy )
    0  PLYMV                ( Move player 0 )
    SWAP                    ( Rotate stick 90 degrees )
    1  PLYMV                ( Move player 1 )
    ?TERMINAL
  UNTIL ;

JOY2  <ret>
```

Notice that when you push the stick up, player zero goes up, but player one moves left. The SWAP instruction exchanges the vertical and horizontal offsets from STICK before moving player one. If we were to take the SWAP out, the players would move identically.

In many applications, it is necessary to know when a player has hit another player or some background image. Fortunately, the Atari computer automatically makes this information available. An entire collection of valFORTH words allows checking of all collisions possible. The most general word is ?COL which simply returns a true flag if anything has hit anything else. Here is an example:

```
: BUMP
  BEGIN
    HITCLR
    0  STICK
    0  PLYMV
    ?COL
    IF
      CR  ." oops!"
    ENDIF
    ?TERMINAL
  UNTIL  ;

BUMP  <ret>
```

Move the player around and watch the results. Every time you hit any letters or player one, the word "oops!" should be printed out. This program is quite simple. First, the HITCLR command is issued which erases any old collision information. If this command were omitted, the first time a collision occurred, "oops!" would be continuously printed out. Next the joystick is read and the player moved. If the player touches anything when moved, the collision registers are set. ?COL reads these registers and leaves a true flag if the player has hit something, and the IF statement will then print out "oops!".

Using other commands found in the glossary, we can tell specifically what the player has hit. For example, the ?PXPF command checks to see if a specific player has hit a playfield, and if so, it returns information indicating which playfield.

Although this discussion was limited to using players, the routines for missiles function similarly and can be found in the following glossary. Two player/missile example programs can be found on your Player/Missile disk. These demonstrate how short player/missile routines can be.

PLAYER/MISSILE GLOSSARY

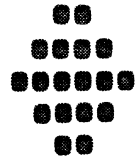
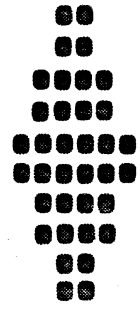
Enabling Player-Missile Graphics

To make use of players and missiles, the video processor must be activated. Players can be several sizes, they can have different overlap priority schemes, and they can have different colors. The following collection of "words" makes this setup task quite simple. Note: Players and missiles are numbered 0 through 3. The fifth player is numbered as four.

(PMINIT) (addr res ---)

The (PMINIT) command (or PMINIT below) must be used to initialize the player missile routines before any other player missile command may be used. (PMINIT) expects both the address of player/missile memory and a 1 or a 2 indicating whether single or double resolution is desired.

NOTE: The difference between single and double resolution is shown graphically below:

Player as defined in memory:	single res on screen:	double res on screen:
00011000		
00111100		
01111110		
00111100		
00011000		

PMINIT (res ---)

The PMINIT command functions identically to the (PMINIT) command above, except that no address need be given. PMINIT calculates an address based on the current graphic mode. It uses the first unused 2K block of memory below the highest free memory (i.e., below the display list). This should only be used while first learning the system, after that, (PMINIT) should be used to optimize memory utilization. Note that the variable PMBAS contains the calculated address upon return.

PMBAS (--- addr)

A variable containing the address of player/missile memory. This value must lie on a 2K boundary if single resolution players are used and on a 1K boundary if double resolution players are used. This is set using the (PMINIT) command and is automatically set by the PMINIT command described above. This value should never be set directly, but can be read at any time.

PLAYERS (ON/OFF ---)

If the flag found on the top of the stack equates to TRUE or ON, then the player/missiles are activated. This does not clear out player missile memory; therefore, the PMCLR command described below is usually used prior to enabling the players and missiles to ensure that no random trash appears on the screen.

If the flag found on the top of the stack equates to FALSE or OFF, then the player/missile graphic mode is de-activated. Turning players off does not clear player-missile memory; therefore, a subsequent ON PLAYERS command would redisplay any previously defined players and missiles. If players are already disabled, the command is ignored.

5THPLY (flag ---)

In many applications it is desirable to combine the four missiles and simulate a fifth player, thus giving five players (numbered 0-4), and no missiles. If the flag on the stack is non-zero, then the fifth player mode will be initiated; otherwise, the missile mode will be re-activated.

Normally, missiles take on the color of their corresponding players; however, when a fifth player is asked for, all missiles take on the common color of playfield #3. In addition, it also allows the fifth player to be treated exactly as any other player would be treated. Bear in mind that although it is called a "fifth" player, its reference number is four (4). The fifth player is "built" with missile zero on the right, and missile three on the left:

|m3|m2|m1|m0| = fifth player

(Note: For convenience, the words ON and OFF have been defined to allow niceties such as:

```
ON 5THPLY
OFF 5THPLY
```

These two words are recognized by all words that require an ON/OFF type indication.)

PLYCLR (pl# ---)

Few applications use all available players. To keep these unused players from displaying trash, they can be cleared of all data by using the PLYCLR command. The PLYCLR command expects the player number on the top of the stack and fills the specified player with zeroes. This command can be used to "turn off" players which are no longer needed.

MSLCLR (ml# ---)

The MSLCLR command is very much like the PLYCLR command, described above, except that it clears the specified missile. In addition, this can be used when the fifth player is activated to erase parts of the fifth player for special effects.

PMCLR (---)

This command clears all players and all missiles. This is generally used just prior to activating the player-missile graphic mode to ensure that no random trash is placed on the video screen. PMCLR expects no values on the stack, nor does it leave any.

MCPLY (F ---)

The MCPLY (Multi-Color Player) command expects one value on the top of the stack. If this value is 0 or OFF, then the multi-color player mode is disabled. If this value is 1 or ON, this command instructs the video processor to logically "or" the bits of the colors of player zero with player one, and also of player two with player three. In other words, when players 0 and 1 overlap (or players 2 and 3), a third color (determined by the colors of the overlapping players) will be assigned to the overlapped region rather than assigning one of the players a higher priority. Since players must be one color, this allows for multi-colored players. For example:

Player 0 Pink color (4)	Player 1 Blue color (8)	MCPlayer Pink/blue (4 OR 8 = green)
	BBBB	BBBB
	BBBBBBBB	BBBBBBBB
PPPPPPPP		PPPPPPPP
PPPPPPPP	BB BB	PGGPPGGP
PPPPPPPP		PPPPPPPP
PP PP		PP PP
PPPP		PPPP

NOTE: The lums of the two players are also OR'd.

PRIOR (n ---)

The PRIOR command expects one value on the top of the stack. This value must be 8, 4, 2, or 1, otherwise unpredictable video displays may occur. PRIOR instructs the video processor as to what has higher priority for a video location on the screen. For example, it will determine whether a plane (a player) will pass in front of a building (a playfield), or whether the plane will pass behind the building. Objects with higher priorities will appear to pass in front of those with lower priorities. The following table shows the available priority settings:

n=8	n=4	n=2	n=1
PF0	PF0	PL0	PL0
PF1	PF1	PL1	PL1
PL0	PF2	PF0	PL2
PL1	PF3*	PF1	PL3
PL2	PL0	PF2	PF0
PL3	PL1	PF3*	PF1
PF2	PL2	PL2	PF2
PF3*	PL3	PL3	PF3*
BAK	BAK	BAK	BAK

* PF3 and PL4 share the same priority

Objects higher on the list will appear to pass in front of objects lower on the list.

CREATING PLAYERS AND MISSILES

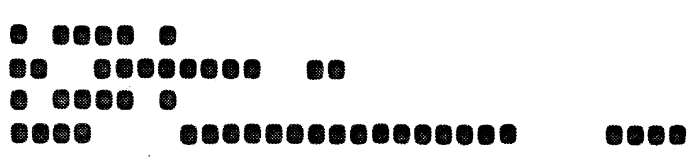
Once the player/missile graphics system has been activated and the priorities set, all that need be done is to create the players themselves. Normally, this would be quite difficult to do; however, using the commands and designing techniques described below, this task is made very simple. There are really only three things to do in the creation of a player: setting the width size, setting the color, and creating the picture.

PLYWID (width pl# ---)

The PLYWID command sets the specified player to the desired width. Players are numbered 0, 1, 2, 3, or in the case of the fifth player, 4. Legal widths are:

image: 10111101

0 = normal width:
1 = double width:
2 = normal width:
3 = quad. width:



Any other value may cause strange results.

MSLWID (size ml# ---)

The MSLWID command is identical to the PLYWID command described above except that it is used to set the size of the missiles. The same size values apply also. The MSLWID command should only be used when in the missile mode (i.e., with the fifth player deactivated).

PMCOL (pl# hue lum ---)

To set the color (hue and lum) of a player, the PMCOL (Player-Missile-Color) command is used. It sets the specified player to the hue and lumina desired. Note that there is no corresponding command to set the colors of missiles as missiles take on the colors of their respective players. To set the color of the 5th player, "pl#" should be 4. If the color words on the valFORTH 1.1 disk are loaded, they can be used to set player colors:

```
0 BLUE 8 PMCOL
```

This sets player #0 to a medium blue color.

BLDPLY (addr len horz vert pl# ---)

The BLDPLY command is probably the most useful of all the commands in this graphic package. It takes an easily predefined picture that resides in memory at address "addr" whose length is "len" and converts it to the specified player "pl#". It then positions the player at the coordinates (horz,vert). The player is then ready to be moved about the screen using the PLYMV command described below.

As an example, a player in the form of an arrow pointing upward will be created, assuming that priorities and such have already been taken care of. Practice has proven that the following method is easiest for creating players:

```
2 BASE  !            ( put into binary mode    )

LABEL PICTURE        ( the image is named PICTURE )
  00011000 C,
  00111100 C,
  01111110 C,
  11011011 C,
  00011000 C,
  00011000 C,
  00011000 C,
  00011000 C,
DECIMAL

1 PMINIT             ( initialize for single resolution )
PICTURE  8  80  40  0  BLDPLY
```

Takes the image at location PICTURE which is 8 bytes long, and builds player #0 at location (80,40).

BLDMSL (addr len horz vert ml# ---)

The BLDPLY command described above does just about everything necessary to create a high-resolution player. The BLDMSL command functions identically to the BLDPLY command except that it is used for setting up missiles (which are in effect just skinny players). The method for creating players can be used for creating missiles as well. Note that if the fifth player mode is activated, the BLDPLY command must be used to create the player.

Building missiles takes a bit more care than building players. Players occupy separate memory, while the four missiles share the same memory. Each missile is two bits wide; all four together are exactly a byte wide. Missile memory is shared with the two lowest bits devoted to missile zero, and the two highest bits devoted to missile three:

| m3 | m3 | m2 | m2 | m1 | m1 | m0 | m0 |

All players with the same shape can use the same image without any problem since they all are a full byte wide. Missiles, however, cannot use the same shape since their images must be ORed into missile memory. This means that the missile images must be in the proper bit columns. For example, the same image for separate missiles could be:

```
11000000   00110000   00001100   00000011
11000000   00110000   00001100   00000011
11000000   00110000   00001100   00000011
  msl#3      msl#2      msl#1      msl#0
```

PUTTING PLAYERS AND MISSILES IN THEIR PLACE

Generally, once a player or missile has been created and put to the video screen, it is moved around. This can be accomplished very easily with the next set of words. Interfacing a movable player with the joystick can improve just about any program which requires input. As a result, it usually gives the program a more professional appearance.

PLYLOC (pl# --- horz vert)

The PLYLOC command (PLaYer LOCation) returns the vertical and horizontal positions of the specified player. This is normally used when a joystick/button setup is being utilized -- i.e., when a joystick is moving a player and the button is used to pinpoint where the player is. A program which draws lines between two dots could use this. The joystick is used to move the player to the desired spot on the screen. Pressing the button tells the program that a selected spot has been made. Once a second spot has been selected, the program then draws a line between them.

MSLLOC (ml# --- horz vert)

The MSLLOC command performs the same function as the PLYLOC command described above except that it is used to find locations of missiles instead of players. Note that using MSLLOC on a fifth player gives meaningless results.

PLYMV (horz vert pl# ---)

The PLaYer MoVe command moves the specified player the direction specified by "vert" and "horz". If "vert" or "horz" is negative, the player is moved up or left respectively, otherwise it is moved down or right unless they happen to be zero in which case nothing happens. The following examples clarify this:

```
 0 -5  0 PLYMV    ( Move player 0 up 5 lines )
-1 -1  3 PLYMV    ( Move player 3 left and up one line )
 3 -1  2 PLYMV    ( Move player 2 up one dot and right 3 )
```

MSLMV (horz vert ml# ---)

The MSLMV is identical in function as the PLYMV command described above except that it is used to move missiles about the video screen.

PLYPUT (horz vert pl# ---)

The PLYPUT command positions player "pl#" to the location (horz,vert) on the video screen.

PLYCHG (addr len pl# ---)

Oftentimes it is necessary to change the image of a player after it has been built. The PLYCHG command allows this to be easily done. The PLYCHG command takes the image with length "len" at address "addr" and assigns it to player "pl#". Note that if the new image is shorter than the previous one, part of the previous image will remain. This can be overcome by executing a PLYCLR command prior to PLYCHG.

PLYSEL (addr # pl# --)

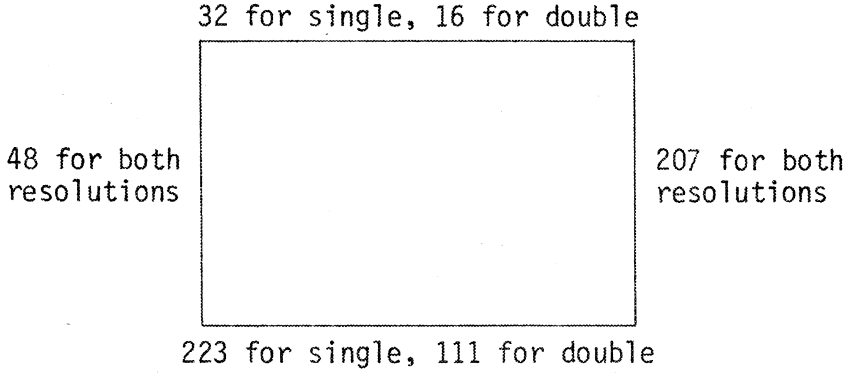
The PLYSEL command is used to select image "#" out of a table of images of the same length and assigns that image to the specified player. PLYSEL is typically used to animate players. An example usage of this can be found in Player/Missile Example #2 found in the directory of the disk.

PLAYER/MISSILE BOUNDARIES

It is often desirable to put limitations on the movements of players and missiles. Boundaries can be set up for each player and missile independently and upon each move command, they will remain within those boundaries. Additionally, a boundary status byte for each player is available for scrutiny at any time. This section explains how this is used.

PLYBND (left right top bottom pl# --)

In most applications, the movements of players are kept within certain boundaries. The PLYBND command frees the user from having to worry about boundary checking. This command expects the player number and all four boundaries. Whenever a PLYMV is then used, the player is always kept within the set boundaries. Also, upon each move a boundary status byte is left in the c-array PLYSTT (see ?PLYSTT below). The edge boundaries of the screen are:



Note that in special cases the boundary checker will fail. If the left boundary is 0 and the player is at the boundary, any move left will not be checked as expected. For example, if it were moved left by one position (-1), the new horizontal position would be -1 or FFFF in hex. Since only 8 bit unsigned comparisons are made, the horizontal position appears to be 255 (FF hex). Post calculating boundary checking turns out to be more useful because it allows any or all edges to be unbounded. If an unbounded player is desired, use this:

```
0  255  0  255  pl#  PLYBND
```

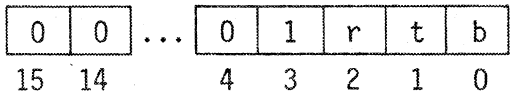
For an example of PLYBND, see the example program found in the directory on screen 170 of your disk.

MSLBND (left right top bottom ml# --)

The MSLBND command is the same as the PLYBND command above, except that it is used for missiles. Upon each move a boundary status byte is left in the array MSLSTT. See ?MSLSTT below.

?BND (--- n)

This command leaves the boundary check status of the last PLYMV or MSLMV performed. The value has the following form:



Only the lower four bits are of use. Each bit represents a different edge. If the bit is set, then the player or missile has attempted to move beyond that boundary. Note that only two of the four bits can be set at any time.

Note:
```
DECIMAL
...
?BND  3  AND
IF  hit-vertical-boundary ENDIF
?BND  12  AND
IF  hit-horizontal-boundary ENDIF
...
```

?PLYSTT (pl# --- val)

Given a player number, returns the boundary check byte of that player. This byte is the status byte for the most recent PLYMV of that player. See ?BND above for the description of the status byte.

?MSLSTT (ml# --- val)

Given a missile number, returns the boundary check byte of that missile. This byte is the status byte for the most recent MSLMV of that missile. See ?BND above for the description of the status byte.

CHECKING FOR INTERACTION BETWEEN PLAYERS

All the commands given so far allow the creation of any player or missile desired. But once that player is on the screen and moving around, it is often necessary to know when two or more objects (players, missiles, and playfields) touch or "crash" into each other. This remaining collection of commands allows checking of all possible "hit" combinations.

?COL (--- f)

The ?COL command is a very general collision detector. It does nothing more than indicate whether two or more objects have "crashed" -- it does not give any indication of what has collided. It leaves a 1 on the stack if a collision has taken place; otherwise it leaves a zero.

?MXPF (ml# --- n)

The ?MXPF command is a much more specific collision detection command. It stands for "?collision of Missile #X with any PlayField". It is used to check if a specific missile has hit any playfield. It returns a zero if no collision has taken place, and leaves an 8, 4, 2, 1, or combinations of these (e.g., 12 = 8+4) if a collision has occurred. Each of these four basic values represents a specific playfield:

3 ?MXPF (Has missile #3 hit any playfields?)

TOS	binary	meaning of val
0	0000	no collisions
1	0001	with pf#0
2	0010	with pf#1
3	0011	with pf#0,1
4	0100	with pf#2
5	0101	with pf#2,0
6	0110	with pf#2,1
7	0111	with pf#2,1,0
8	1000	with pf#3
9	1001	with pf#3,0
10	1010	with pf#3,1
11	1011	with pf#3,1,0
12	1100	with pf#3,2
13	1101	with pf#3,2,0
14	1110	with pf#3,2,1
15	1111	with pf#3,2,1,0

To test for a collision with one specific playfield, use one of the following:

```
1 AND      ( Leaves 1 if collision with pf#0, else 0 )
2 AND      (   "    1       "         "  pf#1,  "   0 )
4 AND      (   "    1       "         "  pf#2,  "   0 )
8 AND      (   "    1       "         "  pf#3,  "   0 )
```

?PXPF (pl# --- n)

The ?PXPF command (?collision of Player #X with any PlayField) behaves in exactly the same manner as the ?MXPF command above except that it tests for collisions with players and playfields instead of missiles and playfields.

?MXPL (ml# --- n)

The ?MXPL command (?collision of Missile #X with any Player) behaves in exactly the same manner as the ?MXPF command above except that it tests for collisions between missiles and players. Note that it is impossible for a missile to collide with a fifth player since it would be, in effect, colliding with itself.

?PXPL (pl# --- n)

The ?PXPL command (?collision of Player #X with any other players) behaves in exactly the same manner as the ?MXPF command above except that it tests for collisions between players. Note that it is impossible for a player to collide with itself.

HITCLR (---)

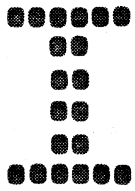
The HITCLR command clears all collision registers. In other words, it sets the collision monitor to a state which indicates that no collisions have occurred.

THE CHARACTER SET EDITOR

Character Sets

Whenever the computer has to display a character on the video screen, it must refer to a table which holds the shape definition for that character. By changing this table, new character sets can be formed.

The shape of a single character in the table (or character set) is made up of 8 bytes of data. A character is one byte wide and 8 bytes tall forming an 8 by 8 bit matrix. If a bit in this matrix is set (1), then a dot will appear on the screen. If a bit is reset (0), nothing is displayed. For example, the letter I could be defined as:



00000000	$00 = 0
01111110	$7E = 126
00011000	$18 = 24
00011000	$18 = 24
00011000	$18 = 24
00011000	$18 = 24
01111110	$7E = 126
00000000	$00 = 0

Thus, the sequence 0, 126, 24, 24, 24, 24, 126, 0, represents the letter I. The entire alphabet is constructed in this fashion. By selectively setting the bit pattern, custom made characters can be formed. This can find many uses. A British character set can be made by changing the one character "#" to the British monetary symbol. Likewise, a Japanese character set could be made by replacing the lowercase characters with Katakana letters.

Another use would be to design special symbol sets. For example, an entire set could be devoted to special mathematical symbols such as plus-minus signs, square-root signs integration signs, or vector signs. (Although this would be of little use in normal operation where character sets cannot be mixed on the same line, using the high resolution text output routines in the Editor/Utilities package. It becomes easy to mix character sets in this fashion.) Assuming the character sets were defined, it would be possible to have a Japanese quotation (in kana of course) embedded within the text of a mathematical explanation of some kind all on the same line!

A final use for custom character sets is for "map-making." Characters can be designed so that they can be pieced togehter to form a picture. An excellent example of this can be found in Cris Crawford's Eastern Front game available through the Atari Program Exchange. When done properly, the final "puzzle" will appear as though it is a complicated high resolution picture.

Now, on to the editor...

The Editor

The following description explains how to use the character editor found on the Player/Missile disk. This editor allows a character set to be designed and then saved on disk for later modification or use. A copy of the standard character has already been saved and can be located through the directory on screen 170.

After loading the character editor, it is executed by typing:

CHAR-EDIT <ret>

The screen has an 8 by 8 grid in the upper-lefthand corner. On the right side there is a command list, and at the bottom, a section is reserved to display the current character set.

The Commands:

I) The joystick

A joystick in port 0 (the leftmost port) is used to move the character cursor (the solid circle) within the 8 by 8 grid. The cursor indicates where the next change to the current character will be made.

II) The button

When pressed, the joystick button will toggle the bit under the character cursor in the 8 by 8 grid. If the bit is set (on), it will be reset. If the bit is reset (off), it will be set. The character will be updated in the character set found at the bottom of the screen.

III) "1" command

By pressing the "1" the current character is cleared in both the grid and in the character set at the bottom of the display. There is no verify prompt for this command.

IV) "2" command

By pressing the "2" key the current character and character set are cleared. User verification is required before any action is taken.

V) "3" command

By pressing the "3" key the current character is saved to disk. User verification is required with a yes/no response. If a yes response is given, a screen number is asked for and the current character set is saved on the specified screen. The current character is not destroyed upon a save.

VI) "4" command

By pressing the "4" key a character set is loading from disk, destroying the current character set. User verification is required with a yes/no response. If a yes response is given, a screen number is asked for and a character set loaded from the specified screen.

VII) "<-- " and "-->" commands

These two arrow keys move the character pointer through the character set to allow modification of any character in the current set.

VIII) Console key

Pressing any console key terminates the edit session and returns control to the FORTH system. The current character set is lost unless it is saved to disk prior to ending the session.

Loading Character Sets

The following three words allow easy use of custom character sets.

CHLOAD (addr scr# cnt ---)

The CHLOAD command takes the first "cnt" characters on screen "scr#" and stores them consecutively starting at address "addr". Each screen (in half-K mode) will only hold 64 character definitions. If "cnt" is greater than 64, CHLOAD will continue loading from the next screen. Many character sets could be loaded at one time by giving a very large "cnt" value. Besides being able to load a full set, the CHLOAD command allows the building of a new set from several other sets.

Note that if a 20 character/line mode is being used, "addr" should lie on a half-K boundary (only upper 7 bits significant). If a 40 character/line mode is being used, "addr" should lie on an 1K boundary (only upper 6 bits significant). Also note that PAD is modified by CHLOAD.

SPLCHR (addr ---)

The SPLCHR commands activates the character set at the address specified.

NMLCHR (---)

The NMLCHR command re-activates the normal character set.

AUDIO-PALETTE -- A SOUND EDITOR

Audio-Palette is a sound editor which generates all possible time-in dependent sounds that the Atari 400/800 microcomputer can produce. Each of the four channels are interfaced to one of the four joystick ports. The joysticks allow the setting of the pitch (horizontal) the distortion (vertical) of their corresponding channel. When the joystick button is pushed, the sound is made. To get a better idea of how this works, load the editor (see screen 170) and type:

AUDED <ret>

The screen should clear and a table of values should appear at the bottom of the display. In the upper lefthand corner of the screen, there should be four numerals (players) overlayed (one for each channel). Each of these players can be moved around the display by using a joystick in the appropriate port.

As a player is moved vertically, the distortion changes. As a player is moved horizontally, the pitch changes. By pressing the button, a sound will be made according to the current frequency (pitch), distortion, volume, and audio control settings. To increase the volume, the up-arrow is used. Any time the up-arrow is pressed, all channels whose corresponding joystick buttons are pressed will have their volumes increased. Likewise, the down-arrow will decrease the volumes.

Each bit of the audio control value performs some function in the sound generator. The bits are numbered 0 to 7. Pressing the keys 0 to 7 will toggle the corresponding bits in the audio control register. For a description of these bit settings, please refer to the explanation of SOUND in the valFORTH 1.1 package.

```
Screen:  30
  0 ( PlyMsl:  arrays and variables)
  1 BASE @
  2 DCX '( ARRAY )( 80 KLOAD )
  3  0 VARIABLE PMBAS
  4  5   CARRAY PLYVRT
  5  5   CARRAY PLYHRZ
  6  5   CARRAY PLYLEN
  7  5    ARRAY PLYADR
  8  4   CARRAY MSLVRT
  9  4   CARRAY MSLHRZ
 10  4   CARRAY MSLLEN
 11  4    ARRAY MSLADR
 12  5    ARRAY PMADR
 13  0 VARIABLE PMLEN
 14  0 VARIABLE PMRES
 15  0 VARIABLE MSLSZ                ==>
```

```
Screen:  31
  0 ( PlyMsl:  arrays and variables)
  1
  2  0 VARIABLE BOUNDS 34 ALLOT
  3  5   CARRAY PLYSTT
  4  4   CARRAY MSLSTT
  5  0 VARIABLE BNDCOL
  6  2 VARIABLE 5THWID
  7
  8  CTABLE 5THDAT
  9  2 C, 4 C, 2 C, 8 C,
 10
 11  HEX
 12
 13  CTABLE MSLDAT
 14  FC C, F3 C, CF C, 3F C,
 15                              -->
```

```
Screen:  32
  0 ( PlyMsl:  [PMINIT]             )
  1
  2 : (PMINIT)      ( addr  res  -- )
  3   SWAP PMBAS ! 1- DUP PMRES !
  4   NOT 10 * 0C OR
  5   22F C@ EF AND OR 22F C!
  6   PMBAS @ 180 PMRES @
  7   NOT 1+ )R
  8   R * + DUP 4 PMADR !
  9   80 R) * )R
 10   R + DUP 0 PMADR !
 11   R + DUP 1 PMADR !
 12   R + DUP 2 PMADR !
 13   R +     3 PMADR !
 14   R) PMLEN ! ;
 15                              ==>
```

```
Screen:  33
  0 ( PlyMsl:  PMINIT  PLAYERS      )
  1
  2 : PMINIT                ( res -- )
  3   2E6 C@ 8 - F8 AND
  4   OVER 1- 4 * + 100 *
  5   SWAP (PMINIT) ;
  6
  7 : PLAYERS                 ( f -- )
  8   IF
  9     PMBAS @ DUP
 10     PMRES @ 1+ (PMINIT)
 11     SP@ 1+ C@ SWAP
 12     DROP D407 C!
 13     SGRCTL @ 3 OR DUP
 14     SGRCTL ! D01D C!
 15   ELSE                         -->
```

```
Screen:  34
  0 ( PlyMsl:  5THPLY               )
  1
  2     SGRCTL @ FC AND
  3     DUP SGRCTL ! D01D C!
  4     22F C@ E3 AND 22F C!
  5     D00D 5 ERASE
  6   ENDIF ;
  7
  8
  9 : 5THPLY                  ( f -- )
 10   26F C@ SWAP
 11   IF 10 OR
 12   ELSE EF AND
 13   ENDIF
 14   26F C! ;
 15                               ==>
```

```
Screen:  35
  0 ( PlyMsl:  PMCLR  PLYCLR        )
  1
  2
  3 : PMCLR                   ( -- )
  4   4 PMADR @
  5   PMLEN @ 5 *
  6   0 FILL ;
  7
  8
  9 : PLYCLR                ( pl# -- )
 10   PMADR @
 11   PMLEN @
 12   0 FILL ;
 13
 14
 15                               -->
```

```
Screen:  36
  0 ( PlyMsl:  MSLCLR PRIOR          )
  1
  2 : MSLCLR                  ( ml# -- )
  3   4 PMADR @ DUP
  4   PMLEN @ + SWAP
  5   DO
  6     DUP MSLDAT C@
  7     I C@ AND I C!
  8   LOOP
  9   DROP ;
 10
 11 : PRIOR                       ( n -- )
 12   26F C@ 0F0 AND
 13   OR 26F C! ;
 14
 15                                   ==>
```

```
Screen:  37
  0 ( PlyMsl:  PLYMV                 )
  1
  2 CODE PLYMV
  3  84 C, N 6 + C, B5 C, 00 C,
  4  0A C, A8 C, B9 C, 0 PMADR 1+ ,
  5  85 C, N 1+ C, B9 C, 0 PMADR ,
  6  85 C, N 1- C,  B9 C, 0 PLYADR ,
  7  85 C, N 2+  C, B9 C, 0 PLYADR
  8  1+  , 85 C, N 3 + C, B4 C, 0 C,
  9  B9 C, 0 PLYLEN , 85 C, N 4 + C,
 10  B9 C, 0 PLYHRZ , 18 C, 75 C,
 11  04 C, D9 C, BOUNDS , B0 C, 5 C,
 12  B9 C, BOUNDS , E6 C, N 6 + C,
 13  06 C, N 6 + C, D9 C, BOUNDS 5 +
 14  , F0 C, 07 C, 90 C, 05 C, B9 C,
 15  BOUNDS 5 + , E6 C, N 6 + C, -->
```

```
Screen:  38
  0 ( PlyMsl:  PLYMV                 )
  1  99 C, 0 PLYHRZ , 95 C, 05 C,
  2  B9 C, 0 PLYVRT , 85 C, N C,
  3  18 C, 75 C, 2 C, 06 C, N 6 + C,
  4  D9 C, BOUNDS A + , B0 C, 05 C,
  5  B9 C, BOUNDS A + , E6 C, N 6 +
  6  C, 6 C, N 6 + C, D9 C, BOUNDS
  7  F + , F0 C, 07 C, 90 C, 05 C,
  8  B9 C, BOUNDS F + , E6 C, N 6 +
  9  C, 99 C, 0 PLYVRT , 95 C, 3 C,
 10  38 C, E5 C, N C, B0 C, 05 C,
 11  A5 C, N C, 38 C, F5 C, 03 C,
 12  95 C, 02 C, C5 C, N 4 + C,
 13  90 C, 02 C, A5 C, N 4 + C,
 14  85 C, N 5 + C,
 15                                   ==>
```

```
Screen:  39
  0 ( PlyMsl:  PLYMV                 )
  1  A5 C, N C, D5 C, 03 C, 90 C,
  2  08 C, 18 C, 65 C, N 4 + C, 38
  3  C, E5 C, N 5 + C, 85 C, N C,
  4  18 C, 65 C, N 1- C, 85 C, N C,
  5  B5 C, 2 C, F0 C, 0B C, A0 C,
  6  00 C, 98 C, 88 C, C8 C, 91 C,
  7   N C, C4 C, N 5 + C, D0 C,
  8  F9 C, B5 C, 00 C, C9 C, 04 C,
  9  D0 C, 14 C, B5 C, 05 C, A0 C,
 10  04 C, HERE 88 C, 30 C, 0A C,
 11  99 C, D004 , 18 C, 6D C, 5THWID
 12  , 4C C, , 4C C, HERE 2 ALLOT
 13  B5 C, 05 C, B4 C, 00 C, 99 C,
 14  D000 , HERE SWAP ! B4 C, 00 C,
 15  A5 C, N 6 + C,                   -->
```

```
Screen:  40
  0 ( PlyMsl:  PLYMV                 )
  1
  2  99 C, 0 PLYSTT , 8D C, BNDCOL ,
  3  B5 C, 3 C, 18 C, 65 C, N 1- C,
  4  85 C, N C, A0 C, 00 C,
  5  B1 C, N 2+ C,
  6  91 C, N C, C8 C, C4 C, N 4 + C,
  7  D0 C, F7 C, E8 C, E8 C,
  8  4C C, POPTWO ,   C;
  9
 10
 11
 12
 13
 14
 15                                   ==>
```

```
Screen:  41
  0 ( PlyMsl:  MSLMV                 )
  1 HEX
  2
  3 CODE MSLMV
  4  84 C, N 6 + C, B5 C, 0 C, 0A C,
  5  A8 C, AD C, 4 PMADR 1+ , 85 C,
  6  N 1+ C, AD C, 4 PMADR , 85 C,
  7  N 1- C, B9 C, 0 MSLADR , 85 C,
  8  N 2+ C, B9 C, 0 MSLADR 1+ ,
  9  85 C, N 3 + C, B4 C, 0 C, B9 C,
 10  0 MSLDAT , 85 C, N 7 + C, B9 C,
 11  0 MSLLEN , 85 C, N 4 + C, B9 C,
 12  0 MSLHRZ , 18 C, 75 C, 04 C,
 13  D9 C, BOUNDS 14 + , B0 C, 5 C,
 14  B9 C, BOUNDS 14 + , E6 C, N 6 +
 15                                   -->
```

```
Screen:  42
  0 ( PlyMsl:  MSLMV                    )
  1
  2  C, 6 C, N 6 + C, D9 C, BOUNDS
  3  18 + , F0 C, 07 C, 90 C,
  4  05 C, B9 C, BOUNDS 18 + ,
  5  E6 C, N 6 + C,
  6  99 C, 0 MSLHRZ , 95 C, 05 C,
  7  B9 C, 0 MSLVRT , 85 C, N  C,
  8  18 C, 75 C, 02 C, 6 C, N 6 + C,
  9  D9 C, BOUNDS 1C + , B0 C, 5 C,
 10  B9 C, BOUNDS 1C + , E6 C, N 6 +
 11  C, 06 C, N 6 + C, D9 C, BOUNDS
 12  20 + , F0 C, 7 C, 90 C, 5 C,
 13  B9 C, BOUNDS 20 + , E6 C, N 6 +
 14  C, 99 C, 0 MSLVRT , 95 C, 3 C,
 15                                 ==>
```

```
Screen:  43
  0 ( PlyMsl:  MSLMV                    )
  1
  2  38 C, E5 C, N C, B0 C, 5 C, A5
  3  C, N C, 38 C, F5 C, 3 C, 95 C,
  4  2 C, C5 C, N 4 + C, 90 C, 2 C,
  5  A5 C, N 4 + C, 85 C, N 5 + C,
  6  A5 C, N C, D5 C, 3 C, 90 C,
  7  8 C, 18 C, 65 C, N 4 + C, 38
  8  C, E5 C, N 5 + C, 85 C, N C,
  9  18 C, 65 C, N 1- C, 85 C, N C,
 10  A0 C, FF C, C8 C, B1 C, N C,
 11  25 C, N 7 + C, 91 C, N C, C4
 12  C, N 5 + C, D0 C, F5 C, B5 C,
 13  5 C, B4 C, 0 C, 99 C, D004 ,
 14
 15                                 -->
```

```
Screen:  44
  0 ( PlyMsl:  MSLMV                    )
  1
  2   B4 C, 0 C, A5 C, N 6 + C, 99
  3   C, 0 MSLSTT , 8D C,
  4   BNDCOL , B5 C, 3 C, 18 C,
  5   65 C, N 1- C, 85 C, N C,
  6   A0 C, 00 C, B1 C, N C,
  7   25 C, N 7 + C, 11 C, N 2+ C,
  8   91 C, N C, C8 C,
  9   C4 C, N 4 + C, D0 C, F3 C, E8
 10   C, E8 C, 4C C, POPTWO , C;
 11
 12
 13
 14
 15                                 ==>
```

```
Screen:  45
  0 ( PlyMsl:  BLDPLY  BLDMSL           )
  1
  2 : BLDPLY       ( a l h v pl# -- )
  3   )R              R PLYVRT C!
  4   R PLYHRZ C!     R PLYLEN C!
  5   R PLYADR  !   ( R PLYCLR )
  6   0 0 R) PLYMV ;
  7
  8 : BLDMSL       ( a l h v pl# -- )
  9   )R              R MSLVRT C!
 10   R MSLHRZ C!     R MSLLEN C!
 11   R MSLADR  !   ( R MSLCLR )
 12   0 0 R) MSLMV ;
 13
 14
 15                                 -->
```

```
Screen:  46
  0 ( PlyMsl:  PLYCHG PLYSEL PLYPUT)
  1
  2 : PLYCHG          ( a len pl# -- )
  3   )R R PLYLEN C!
  4   R PLYADR !
  5   0 0 R) PLYMV ;
  6
  7 : PLYSEL            ( a # pl# -- )
  8   )R R PLYLEN C@ * +
  9   R PLYLEN C@ R) PLYCHG ;
 10
 11 : PLYPUT            ( h v pl# -- )
 12   )R R PLYVRT C@ -
 13   SWAP R PLYHRZ C@ -
 14   SWAP R) PLYMV ;
 15                                 ==>
```

```
Screen:  47
  0 ( PlyMsl:  PLYWID                   )
  1
  2 CODE PLYWID
  3   B5 C, 00 C, C9 C, 04 C, F0 C,
  4   09 C, A8 C, B5 C, 02 C, 99 C,
  5   D008 , 4C C, HERE 2 ALLOT
  6   A8 C, A0 C, 04 C, 0A C, 0A C,
  7   15 C, 02 C, 88 C, D0 C, F9 C,
  8   8D C, MSLSZ , 8D C, D00C ,
  9   B4 C, 02 C, B9 C, 0 5THDAT ,
 10   85 C,  N C, 8D C, 5THWID ,
 11   AD C, 4 PLYHRZ , A0 C, 04 C,
 12   HERE 88 C, 30 C, 09 C, 99 C,
 13   D004 , 18 C, 65 C, N C, 4C C,
 14   , HERE SWAP ! 4C C, POPTWO ,
 15 C;                              -->
```

```
Screen:  48
  0 ( PlyMsl:  MSLWID                    )
  1
  2 CODE MSLWID
  3   B4 C, 00 C, B9 C, 0 MSLDAT ,
  4   2D C, MSLSZ , HERE
  5   88 C, 30 C, 7 C, 16 C, 02 C,
  6   16 C, 02 C, 4C C, , 15 C,
  7   02 C, 8D C, MSLSZ , 8D C,
  8   D00C , 4C C, POPTWO ,
  9 C;
 10
 11
 12
 13
 14
 15                                   ==>
```

```
Screen:  49
  0 ( PlyMsl:  PLYLOC MSLLOC MCPLY )
  1
  2 CODE PLYLOC          ( pl# -- h v )
  3  94 C, 01 C, B4 C, 0 C,
  4  B9 C, 0 PLYHRZ , 95 C, 0 C,
  5  B9 C, 0 PLYVRT , 4C C, PUSH0A ,
  6
  7 CODE MSLLOC          ( ml# -- h v )
  8  94 C, 01 C, B4 C, 0 C,
  9  B9 C, 0 MSLHRZ , 95 C, 0 C,
 10  B9 C, 0 MSLVRT , 4C C, PUSH0A ,
 11
 12 : MCPLY                      ( f -- )
 13   26F C@ SWAP
 14   IF 20 OR ELSE DF AND ENDIF
 15   26F C! ;                          -->
```

```
Screen:  50
  0 ( PlyMsl:  ?COL HITCLR ?MXPF...)
  1
  2 CODE ?COL                     ( -- f )
  3  CA C, CA C, 98 C, A0 C, 0F C,
  4  19 C, D000 , 88 C, 10 C, FA C,
  5  C8 C, 94 C, 01 C, 95 C, 00 C,
  6  4C C, ' 0# ( CFA @ ) ,
  7 C;
  8
  9 CODE ?MXPF             ( ml# -- n )
 10   B4 C, 00 C,  B9 C, D000 ,
 11   4C C, PUT0A ,  C;
 12
 13 CODE ?PXPF             ( pl# -- n )
 14   B4 C, 00 C,  B9 C, D004 ,
 15   4C C, PUT0A ,  C;              ==>
```

```
Screen:  51
  0 ( PlyMsl:  ?MXPL ?PXPL PLYBND  )
  1
  2 CODE ?MXPL             ( ml# -- n )
  3   B4 C, 00 C,  B9 C, D008 ,
  4   4C C, PUT0A ,  C;
  5
  6 CODE ?PXPL             ( pl# -- n )
  7   B4 C, 00 C,  B9 C, D00C ,
  8   4C C, PUT0A ,  C;
  9
 10 CODE HITCLR                   ( -- )
 11   8C C, D01E , 4C C, NEXT , C;
 12
 13 CODE ?BND              ( x1# -- n )
 14   AD C, BNDCOL ,
 15   4C C, PUSH0A ,  C;             -->
```

```
Screen:  52
  0 ( PlyMsl:  MSLBND  ?BND        )
  1
  2 CODE ?PLYSTT           ( pl# -- n )
  3   B4 C, 00 C, B9 C, 0 PLYSTT ,
  4   4C C, PUT0A ,  C;
  5
  6
  7 CODE ?MSLSTT           ( ml# -- n )
  8   B4 C, 00 C, B9 C, 0 MSLSTT ,
  9   4C C, PUT0A ,  C;
 10
 11 : PLYBND       ( l r t b pl# -- )
 12   )R 4 ROLL )R
 13   (ROT SWAP R) R)
 14   BOUNDS + 14 0+S
 15   DO I C! 5 /LOOP ;              ==>
```

```
Screen:  53
  0 ( PlyMsl:  PMCOL               )
  1
  2 : MSLBND       ( l r t b ml# -- )
  3   )R 4 ROLL )R
  4   (ROT SWAP R) R)
  5   BOUNDS + 14 + 10 0+S
  6   DO I C! 4 /LOOP ;
  7
  8 : PMCOL        ( pl# col lum -- )
  9   SWAP 10 * +
 10   SWAP DUP 4 =
 11   IF
 12     DROP 2C7 C!
 13   ELSE
 14     2C0 + C!
 15   ENDIF ;                        -->
```

```
Screen:  54
  0 ( PlyMsl:  initialization      )
  1
  2 DCX
  3
  4 BOUNDS       36   0 FILL
  5 BOUNDS  5 +  5 255 FILL
  6 BOUNDS 15 +  5 255 FILL
  7 BOUNDS 24 +  4 255 FILL
  8 BOUNDS 32 +  4 255 FILL
  9
 10 0 PLYSTT 5 ERASE
 11 0 MSLSTT 4 ERASE
 12
 13 1 PMINIT     ( Set up defaults )
 14
 15 BASE !
```

```
Screen:  55
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  56
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  57
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  58
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  59
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  60
  0 ( Audio Editor                    )
  1
  2 BASE @ DCX
  3
  4 '( PLYMV )( 15 KLOAD )
  5 '( SOUND )( 83 KLOAD )
  6 '( STICK )( 84 KLOAD )
  7
  8
  9 VOCABULARY AUDPAL IMMEDIATE
 10 AUDPAL DEFINITIONS
 11
 12 4 CARRAY PIT
 13 4 CARRAY VOL
 14 4 CARRAY DST
 15 0 VARIABLE ACTL                ==>
```

```
Screen:  61
  0 ( Audio Editor                    )
  1
  2 HEX
  3  CTABLE TBL
  4  32 C, 1F C, 1E C, 1A C, 18 C,
  5  1D C, 1B C, 33 C, 0F C, 0E C,
  6 DCX
  7
  8 : WPIT                  ( pl# -- )
  9   10 OVER 20 + POS.  PIT C@
 10   3 .R ;
 11
 12 : WDST                  (  pl# -- )
 13   16 OVER 20 + POS.  DST C@
 14   2 .R ;
 15                                 -->
```

```
Screen:  62
  0 ( Audio Editor                    )
  1
  2 : WVOL                  ( pl# -- )
  3   20 OVER 20 +
  4   POS. VOL C@ 2 .R ;
  5
  6 : WACTL                       ( -- )
  7   28 21 POS. BASE C@ ACTL C@
  8   DUP DUP 3 .R 2 BASE C!
  9   26 22 POS. 0
 10   (# # # # # # # # # #) TYPE
 11   FILTER! BASE C! ;
 12
 13
 14
 15                                 ==>
```

```
Screen:  63
  0 ( Audio Editor                    )
  1 HEX
  2 : SETP                        ( -- )
  3   2 PMINIT PMCLR 1 PRIOR
  4   0  3 ( RDORNG ) 6  PMCOL
  5   1  8 ( BLUE   ) 6  PMCOL
  6   2  4 ( PINK   ) 8  PMCOL
  7   3  1 ( GOLD   ) 6  PMCOL
  8   4 0
  9   DO
 10    1 I PLYWID
 11    E080 I 8 * + 8 37 15 I
 12    BLDPLY
 13   LOOP
 14   ON PLAYERS ;
 15 DCX                             -->
```

```
Screen:  64
  0 ( Audio Editor                    )
  1
  2 : INIT                        ( -- )
  3   0 GR. 1 752 C! CLS 3 19 POS.
  4   ." Chan  Freq  Dist "
  5   ." Vol    AUDCTL"
  6   4 0
  7   DO
  8     8 I VOL C!
  9     0 I PIT C!
 10     0 I DST C!
 11     CR I 3 SPACES . I WPIT
 12     I WDST I WVOL
 13   LOOP
 14   0 ACTL ! WACTL SETP ;
 15                                 ==>
```

```
Screen:  65
  0 ( Audio Editor                    )
  1
  2 : SND                   ( pl# f -- )
  3   IF
  4    )R R R PIT C@ R DST C@
  5    R) VOL C@ SOUND
  6   ELSE
  7    XSND
  8   ENDIF ;
  9 HEX
 10 CODE DIG                    ( n -- n )
 11   B5 C, 00 C, 94 C, 00 C,
 12   94 C, 01 C, 38 C, A8 C,
 13   36 C, 00 C, 36 C, 01 C,
 14   88 C, D0 C, F9 C, 4C C,
 15   NEXT , C; DCX                 -->
```

```
Screen:  66
  0 ( Audio Editor                    )
  1
  2 : VOLUPD                    ( n -- )
  3   4 0
  4   DO
  5    I STRIG
  6    IF
  7     DUP I VOL C@ + 0 MAX 15 MIN
  8     I VOL C! I WVOL
  9    ENDIF
 10   LOOP
 11   DROP ;
 12
 13
 14
 15                                  ==>
```

```
Screen:  67
  0 ( Audio Editor                    )
  1
  2 : AKEY            ( -- n tf / ff )
  3   0 764 C@ DUP 255 <>
  4   IF
  5    255 764 C!
  6    10 0
  7    DO
  8     DUP I TBL C@ =
  9     IF
 10      DROP NOT I SWAP 0 LEAVE
 11     ENDIF
 12    LOOP
 13   ENDIF
 14   DROP ;
 15                                  -->
```

```
Screen:  68
  0 ( Audio Editor                    )
  1
  2 : ?AKEY                     ( -- )
  3   AKEY
  4   IF
  5    DUP 8 (
  6    IF
  7     ACTL C@ SWAP 1+ DIG XOR
  8     ACTL C! WACTL
  9    ELSE
 10     9 = 2* 1- VOLUPD
 11    ENDIF
 12   ENDIF ;
 13
 14
 15                                  ==>
```

```
Screen:  69
  0 ( Audio Editor                    )
  1
  2 : PDADJ        ( hrz vrt pl# -- )
  3   )R -DUP
  4   IF 2* R DST C@ +
  5    0 MAX 14 MIN R DST C!
  6    R WDST
  7   ENDIF
  8   -DUP
  9   IF I PIT C@ +
 10    0 MAX 255 MIN R PIT C!
 11    R WPIT
 12   ENDIF
 13   R> DROP ;
 14
 15                                  -->
```

```
Screen:  70
  0 ( Audio Editor                    )
  1
  2 : DIGMV                   ( pl# -- )
  3   )R R PIT C@ 2/ 55 +
  4   R DST C@ 4 * 21 +
  5   R> PLYPUT ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15                                  ==>
```

```
Screen:  71
  0 ( Audio Editor      AUDED         )
  1
  2 FORTH DEFINITIONS
  3
  4 : AUDED                         ( -- )
  5   AUDPAL INIT
  6   BEGIN 4 0
  7    DO
  8     I STICK I PDADJ
  9     I DIGMV I I STRIG SND
 10    LOOP
 11    ?AKEY ?TERMINAL
 12   UNTIL
 13   OFF PLAYERS 0 752 C!
 14   0 0 POS. XSND4 ;
 15 BASE !  FORTH
```

Screen: 72
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 73
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 74
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 75
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 76
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 77
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 78
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 79
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 80
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 81
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 82
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 83
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 84
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 85
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 86
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 87
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 88
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 89
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```
Screen:  90
  0 ( Charedit: var defs                )
  1 BASE @ DCX
  2 '( POS. )( : POS. 84 C! 85 ! ; )
  3
  4 '( STICK )( 84 KLOAD )
  5
  6 VOCABULARY CHREDT IMMEDIATE
  7 CHREDT DEFINITIONS
  8
  9  0 VARIABLE HORZ
 10  0 VARIABLE VERT
 11  0 VARIABLE CHAR#
 12  0 VARIABLE CURLOC
 13  0 VARIABLE DEFLOC
 14  0 VARIABLE TPTR
 15  0 VARIABLE CSET-LOC          ==>
```

```
Screen:  91
  0 ( Charedit                          )
  1
  2 : POSCUR                ( n n -- )
  3   SWAP CURLOC @
  4   DUP C@ 84 -
  5   SWAP C! 40 * + 203 +
  6   88 @ + DUP C@
  7   84 + OVER C!
  8   CURLOC ! ;
  9
 10 : CLICK                      ( -- )
 11   0 53279 C!
 12   8 53279 C! ;
 13
 14
 15                                 -->
```

```
Screen:  92
  0 ( Charedit                          )
  1
  2 HEX
  3 : ANTIC                     ( f -- )
  4   22F C@ SWAP
  5   IF 20 OR ELSE DF AND ENDIF
  6   22F C! ;
  7
  8 CODE CHSB0                ( b -- n )
  9  B4 C, 00 C, C8 C, A9 C, 00 C,
 10  95 C, 00 C, 95 C, 01 C, 38 C,
 11  36 C, 00 C, 36 C, 01 C, 18 C,
 12  88 C, D0 C, F8 C, 4C C, NEXT ,
 13 C;
 14 : CHSB1                 ( n b -- f )
 15   CHSB0 AND 0# ; DCX          ==>
```

```
Screen:  93
  0 ( Charedit                          )
  1 '( NFLG --> )(  )
  2
  3 0 VARIABLE NFLG
  4
  5 : -NUMBER             ( addr -- d )
  6   BEGIN DUP C@ BL = DUP + NOT
  7   UNTIL 0 NFLG ! 0 0 ROT DUP 1+
  8   C@ 45 = DUP >R + -1
  9   BEGIN DPL ! (NUMBER) DUP C@
 10   DUP BL <> SWAP 0# AND
 11   WHILE DUP C@ 46 - NFLG !
 12   0 REPEAT DROP R> IF DMINUS
 13   ENDIF NFLG @ IF 2DROP ENDIF
 14   NFLG @ NOT NFLG ! ;
 15                                 -->
```

```
Screen:  94
  0 ( Charedit                          )
  1
  2 : DSPCHR                      ( -- )
  3   88 @ 203 + CURLOC ! DUP 320 +
  4   SWAP
  5   DO
  6    I 8 0 DO
  7     0 OVER C@ 7 I - CHSB1
  8     IF 128 + ENDIF
  9     CURLOC @ C! 1 CURLOC +!
 10    LOOP
 11    DROP 32 CURLOC +! 40
 12   +LOOP 0 0 VERT ! HORZ ! 88 @
 13   203 + DUP DUP CURLOC ! C@
 14   84 + SWAP C! ;
 15                                 ==>
```

```
Screen:  95
  0 ( Charedit                          )
  1
  2 : GRAFC                       ( -- n )
  3   88 @ 882 + ;
  4
  5 : GR8                         ( -- n )
  6   88 @ 802 + ;
  7
  8 : SCR/W                 ( n n n -- )
  9   SWAP B/SCR * OFFSET @ +
 10   DUP 4 + SWAP
 11   DO
 12    2DUP I SWAP R/W
 13    SWAP 128 + SWAP
 14   LOOP
 15   2DROP ;                       -->
```

```
Screen:  96
 0 ( Charedit                          )
 1 HEX
 2 CODE CHSB2                  ( n -- n )
 3   B5 C, 00 C, 94 C, 00 C,
 4   94 C, 01 C, 38 C, A8 C,
 5   36 C, 00 C, 36 C, 01 C,
 6   88 C, D0 C, F9 C, 4C C,
 7   NEXT , C;
 8 DCX
 9
10 : MPTRR                     ( -- )
11   TPTR @ 0 OVER C! 1+ DUP
12   GR8 2- 33 + U>
13   IF 32 - ENDIF
14   DUP TPTR ! 93 SWAP C! CLICK ;
15                                 ==>
```

```
Screen:  97
 0 ( Charedit                          )
 1
 2 : MPTRL                      ( -- )
 3   TPTR @ 0 OVER C! 1-
 4   DUP GR8 U<
 5   IF
 6    32 +
 7   ENDIF
 8   DUP TPTR !
 9   93 SWAP C!
10   CLICK ;
11
12
13
14
15                                 -->
```

```
Screen:  98
 0 ( Charedit                          )
 1
 2 HEX
 3 : DBMAKE                     ( -- )
 4   OFF ANTIC 58 @ 300 - DUP
 5   58 ! FF00 AND DUP 230 !
 6   DUP 3 70 FILL
 7   3 + DUP 42 SWAP C!
 8   1+   DUP 58  @ SWAP !
 9   2+   DUP 15 2 FILL
10   15 + DUP 12 F FILL
11   12 + DUP 41 SWAP C!
12   1  + 230 @ SWAP !
13   ON ANTIC ;
14 DCX
15                                 ==>
```

```
Screen:  99
 0 ( Charedit                          )
 1
 2 : PTCST                  ( scr# -- )
 3   PAD CSET-LOC !
 4   GRAFC DUP 320 + SWAP
 5   2 0 DO
 6    32 0 DO
 7     DUP DUP 320 + SWAP DO
 8      I C@ CSET-LOC @ C!
 9      1 CSET-LOC +!
10     40 /LOOP
11    1+ LOOP
12    DROP
13   LOOP
14   PAD SWAP 0 SCR/W ;
15                                 -->
```

```
Screen: 100
 0 ( Charedit                          )
 1
 2 : GTCST                  ( scr# -- )
 3   GRAFC PAD ROT 1 SCR/W
 4   PAD CSET-LOC ! 2 0
 5   DO
 6    32 0 DO
 7     DUP DUP 320 + SWAP DO
 8      CSET-LOC @ C@ I C!
 9      1 CSET-LOC +!
10     40 /LOOP
11    1+ LOOP
12   288 + LOOP DROP GRAFC DUP
13   DEFLOC ! DSPCHR 0 CHAR# !
14   GR8 DUP 0 TPTR @ C! 12 14 POS.
15   0 . 93 SWAP C! TPTR ! ;       ==>
```

```
Screen: 101
 0 ( Charedit                          )
 1
 2 : GETSCR                 ( -- scr# )
 3   BEGIN
 4    18 14 POS. ." Screen #: "
 5    PAD  5 EXPECT PAD 1- -NUMBER
 6    DROP 128 17 C! 1 752 C!
 7    18 14 POS. 16 SPACES NFLG @
 8    IF
 9     DUP 1 ( OVER 179 ) OR
10     ?1K IF OVER 89 ) OR ENDIF
11     IF DROP 0 ELSE 1 ENDIF
12    ELSE DROP 0
13    ENDIF
14   UNTIL
15   DUP 13 15 POS. 3 .R ;         -->
```

```
Screen: 102
  0 ( Charedit                                )
  1
  2 : VFIO                          ( -- f )
  3   KEY 89 = 18 14 POS.
  4   18 SPACES ;
  5
  6 : SVCST                           ( -- )
  7   18 14 POS. ." Save this set?"
  8   VFIO
  9   IF GETSCR PTCST ENDIF ;
 10
 11 : LDCST                           ( -- )
 12   18 14 POS. ." Load new set?"
 13   VFIO
 14   IF GETSCR GTCST ENDIF ;
 15                                      ==>
```

```
Screen: 103
  0 ( Charedit                                )
  1
  2 : MVRHT                           ( -- )
  3   CHAR# @ DUP 63 <>
  4   IF
  5    31 =
  6    IF 289 ELSE 1 ENDIF
  7    DEFLOC +!
  8    1 CHAR# +! DEFLOC
  9    @ DSPCHR MPTRR
 10    12 14 POS.
 11    CHAR# ?
 12   ELSE
 13    DROP
 14   ENDIF ;
 15                                      -->
```

```
Screen: 104
  0 ( Charedit                                )
  1
  2 : MVLFT                           ( -- )
  3   CHAR# @ -DUP
  4   IF
  5    32 =
  6    IF -289 ELSE -1 ENDIF
  7    DEFLOC +! -1 CHAR# +!
  8    DEFLOC @ DSPCHR MPTRL
  9    12 14 POS. CHAR# ?
 10   ENDIF ;
 11
 12
 13
 14
 15                                      ==>
```

```
Screen: 105
  0 ( Charedit                                )
  1
  2 : CLRCHR                          ( -- )
  3   DEFLOC @  8 0
  4   DO DUP I 40 * + 0 SWAP C! LOOP
  5   DROP 88 @ 203 +   8 0
  6   DO
  7     DUP I 40 * +   8 0
  8     DO
  9       DUP I + 0 SWAP C!
 10     LOOP DROP
 11   LOOP DROP
 12   0 VERT ! 0 HORZ !
 13   88 @ 203 + DUP C@
 14   84 + SWAP DUP
 15   CURLOC ! C! ;                      -->
```

```
Screen: 106
  0 ( Charedit                                )
  1
  2 : CLRCST                          ( -- )
  3   18 14 POS. ." Clear this set?"
  4   KEY 89 =
  5   IF
  6    GRAFC DUP DUP 680 + SWAP
  7    DO
  8     0 I C!
  9    LOOP
 10    CLRCHR 0 CHAR# ! DEFLOC !
 11    12 14 POS. CHAR# ?
 12    GR8 0 TPTR @ C! 93 OVER
 13    C! TPTR !
 14   ENDIF
 15   18 14 POS. 15 SPACES ;             ==>
```

```
Screen: 107
  0 ( Charedit                                )
  1
  2 HEX
  3
  4 : CKOPT                           ( -- )
  5   2FC C@ FF 2FC C!
  6   DUP 1F = IF CLRCHR ENDIF
  7   DUP 1E = IF CLRCST ENDIF
  8   DUP 18 = IF LDCST  ENDIF
  9   DUP 1A = IF SVCST  ENDIF
 10   DUP 06 = IF MVLFT  ENDIF
 11       07 = IF MVRHT  ENDIF ;
 12
 13
 14
 15 DCX                                  -->
```

```
Screen: 108
  0 ( Charedit                          )
  1
  2 : CKBTN                         ( -- )
  3   644 C@ NOT
  4   IF
  5    CLICK
  6    CURLOC @ DUP C@ 8 CHSB2 XOR
  7    SWAP C! DEFLOC @ VERT @
  8    40 * + DUP C@ 7 HORZ @
  9    - 1+ CHSB2 XOR SWAP C!
 10    2000 0 DO LOOP
 11   ENDIF ;
 12
 13
 14
 15                                  ==>
```

```
Screen: 109
  0 ( Charedit                          )
  1
  2 : CKSTK                         ( -- )
  3   0 STICK 2DUP OR
  4   IF
  5    VERT @ + 0 MAX 7 MIN VERT !
  6    HORZ @ + 0 MAX 7 MIN HORZ !
  7    VERT @ HORZ @ POSCUR
  8    2000 0 DO LOOP
  9   ELSE
 10    2DROP
 11   ENDIF ;
 12
 13 : CHECK                        ( -- )
 14   CKSTK CKBTN CKOPT ;
 15                                  -->
```

```
Screen: 110
  0 ( Charedit                          )
  1
  2 : STPSCR                        ( -- )
  3   CR 4 SPACES
  4   ." * * * CHARACTER-EDIT * * *"
  5   CR CR CR ."  01234567" CR
  6   8 0 DO I . CR LOOP
  7   18 4 POS.
  8   ."      Options:"
  9   18 6 POS.
 10   ." (1) Clear Character"
 11   18 8 POS.
 12   ." (2) Clear this set"
 13   18 10 POS.
 14   ." (3) Save this set"
 15                                  ==>
```

```
Screen: 111
  0 ( Charedit                          )
  1
  2   18 12 POS.
  3   ." (4) Load a new set"
  4   2 14 POS. ." Character 0"
  5   2 15 POS. ." Load/Save: "
  6   2 17 POS.
  7   ." Use '" 30 SPEMIT
  8   ." ' and '" 31 SPEMIT ." ' to"
  9   CR
 10   ." through the character set."
 11   0 0 POS. ;
 12
 13
 14
 15                                  -->
```

```
Screen: 112
  0 ( Charedit                          )
  1
  2 FORTH DEFINITIONS
  3
  4 : CHAR-EDIT                     ( -- )
  5   CHREDT ( enter vocabulary )
  6   0 GR. 1 752 C!
  7   CLS DBMAKE
  8   88 @ 1300 ERASE
  9   GRAFC DEFLOC !
 10   GR8 DUP TPTR !
 11   93 SWAP C!
 12   STPSCR
 13   88 @ 203 + DUP CURLOC !
 14   84 SWAP C!
 15                                  ==>
```

```
Screen: 113
  0 ( Charedit                          )
  1
  2   0 HORZ !
  3   0 VERT !
  4   0 CHAR# !
  5
  6   DCX
  7   BEGIN
  8     CHECK
  9     1 752 C! 128 17 C!
 10     ?TERMINAL
 11   UNTIL
 12   0 GR. ;
 13
 14 BASE !  FORTH
 15
```

Screen: 114
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 115
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 116
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 117
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 118
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 119
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```
Screen: 120
  0 ( Character words:  CHLOAD      )
  1
  2 BASE @ DCX
  3
  4 : CHLOAD     ( addr scr# cnt -- )
  5   8 * DUP <ROT
  6   128 /MOD SWAP 0# +
  7   >R B/SCR * R> 0
  8   DO
  9     PAD 128 I * +
 10     OVER I + 1 R/W
 11   LOOP
 12   DROP
 13   PAD <ROT CMOVE ;
 14
 15                                ==>
```

```
Screen: 121
  0 ( Character words:  NML/SPLCHR )
  1
  2
  3 : SPLCHR              ( CHBAS -- )
  4   SP@ 1+ C@
  5   SWAP DROP 756 C! ;
  6
  7
  8 : NMLCHR                    ( -- )
  9   57344 SPLCHR  ;
 10
 11
 12 BASE !
 13
 14
 15
```

```
Screen: 122
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 123
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 124
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 125
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 126
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 127
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 128
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 129
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 130
  0
  1
  2
  3
  4
  5
  6
  7    ( Standard Character set )
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 131
  0
  1
  2
  3
  4
  5
  6
  7    ( Standard Character set )
  8
  9
 10
 11
 12
 13
 14
 15
```

Screen: 132
 0
 1
 2
 3
 4
 5
 6
 7 (PM example #2 ship images)
 8
 9
10
11
12
13
14
15

Screen: 133
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15

Screen: 134
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15

Screen: 135
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15

Screen: 136
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15

Screen: 137
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15

```
Screen: 138
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 139
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 140
 0 ( Player/Missile example 1      )
 1 '( PLYMV )( 15 KLOAD )
 2 BASE @ 2 BASE !
 3
 4 1 VARIABLE HBALL
 5 1 VARIABLE VBALL
 6
 7 LABEL IMAGE
 8   011100 C,
 9   111110 C,
10   111110 C,
11   111110 C,    ( A BIG BALL )
12   111110 C,
13   111110 C,
14   011100 C,
15                    DECIMAL    ==>
```

```
Screen: 141
 0 ( Player/Missile example 1      )
 1
 2 : BOP 0 53279 C! 8 53279 C! ;
 3
 4 : MOVE-BALL
 5   BEGIN
 6     HBALL @ VBALL @ 0 PLYMV
 7     0 PLYSTT C@ DUP 3 AND
 8     IF VBALL @ MINUS VBALL ! BOP
 9     ENDIF
10     3 >
11     IF HBALL @ MINUS HBALL ! BOP
12     ENDIF
13     50 0 DO LOOP   ( Wait... )
14     ?TERMINAL
15   UNTIL ;                      -->
```

```
Screen: 142
 0 ( Player/Missile example 1      )
 1
 2 : BOUNCE
 3   CLS
 4   1 PMINIT
 5   PMCLR
 6   1 PRIOR
 7   ON PLAYERS
 8   47 200 32 217 0 PLYBND
 9   0 9 ( BLUE ) 8 PMCOL
10   IMAGE 7 100 75 0 BLDPLY
11
12   ." Press START to stop... "
13   MOVE-BALL
14   OFF PLAYERS ;
15                              BASE !
```

```
Screen: 143
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 144
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 145
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 146
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 147
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 148
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 149
```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 150
  0 ( Player/Missile example 2      )
  1 BASE @ DCX
  2 '( CHLOAD )( 60 KLOAD )
  3 '( PLYMV  )( 15 KLOAD )
  4 '( STICK  )( 84 KLOAD )
  5 : FLY
  6   BEGIN
  7     75 0 DO LOOP      ( wait )
  8     PAD               ( addr )
  9     0 PLYLOC SWAP DROP
 10     8 / 11 SWAP -
 11     11 MIN 0 MAX      ( image# )
 12     0 PLYSEL          ( pl#0 )
 13     0 STICK 0 PLYMV
 14     ?TERMINAL
 15   UNTIL ;                      ==>
```

```
Screen: 151
  0 ( Player/Missile example 2      )
  1
  2 : SHIP
  3   2 PMINIT
  4   1 PRIOR
  5   PMCLR
  6   0 9 ( BLUE ) 8 PMCOL
  7   PAD 132 15 CHLOAD
  8   PAD 8 50 50 0 BLDPLY
  9   50 200 10 110 0 PLYBND
 10   CLS
 11   ." Move player with stick 0."
 12   CR
 13   ." Press START to stop... "
 14   ON PLAYERS FLY OFF PLAYERS ;
 15 BASE !                         -->
```

```
Screen: 152
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 153
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 154
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 155
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 156
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 157
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 158
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 159
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 160
 0 ( Utils:  CARRAY  ARRAY          )
 1 BASE @ HEX
 2 : CARRAY          ( cccc, n -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4     ALLOT
 5   ;CODE CA C, CA C, 18 C,
 6   A5 C, W C,  69 C, 02 C, 95 C,
 7   00 C, 98 C, 65 C, W 1+ C,
 8   95 C, 01 C, 4C C,
 9   ' + ( CFA @ ) , C;
10
11 : ARRAY          ( cccc, n -- )
12   CREATE SMUDGE ( cccc: n -- a )
13     2* ALLOT
14   ;CODE 16 C, 00 C, 36 C, 01 C,
15    4C C, ' CARRAY 08 + , C;  ==>
```

```
Screen: 161
 0 ( Utils:  CTABLE  TABLE          )
 1
 2 : CTABLE          ( cccc, -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4   ;CODE
 5   4C C, ' CARRAY 08 + , C;
 6
 7 : TABLE           ( cccc, -- )
 8   CREATE SMUDGE ( cccc: n -- a )
 9   ;CODE
10   4C C, ' ARRAY 0A + , C;
11
12
13
14
15                                 -->
```

```
Screen: 162
  0 ( Utils:  2CARRAY  2ARRAY       )
  1
  2 : 2CARRAY       ( cccc, n n -- )
  3   <BUILDS       ( cccc: n n -- a )
  4    SWAP DUP , * ALLOT
  5   DOES>
  6     DUP >R @ * + R> + 2+ ;
  7
  8 : 2ARRAY        ( cccc, n n -- )
  9   <BUILDS       ( cccc: n n -- a )
 10    SWAP DUP , * 2* ALLOT
 11   DOES>
 12     DUP >R @ * + 2* R> + 2+ ;
 13
 14
 15                                ==>
```

```
Screen: 163
  0 ( Utils:  XC!  X!               )
  1
  2 : XC!     ( n0...nm cnt addr -- )
  3   OVER 1- + >R 0
  4   DO J I - C!
  5   LOOP R> DROP ;
  6
  7 : X!      ( n0...nm cnt addr -- )
  8   OVER 1- 2* + >R 0
  9   DO J I 2* - !
 10   LOOP R> DROP ;
 11
 12 ( Caution: Remember limitation
 13 ( on stack size of 30 values
 14 ( because of OS conflict. )
 15                                -->
```

```
Screen: 164
  0 ( Utils:  CVECTOR  VECTOR       )
  1
  2 : CVECTOR         ( cccc, cnt -- )
  3   CREATE SMUDGE ( cccc: n -- a )
  4   HERE OVER ALLOT XC!
  5   ;CODE
  6   4C C, ' CARRAY 08 + , C;
  7
  8 : VECTOR          ( cccc, cnt -- )
  9   CREATE SMUDGE ( cccc: n -- a )
 10   HERE OVER 2* ALLOT X!
 11   ;CODE
 12   4C C, ' ARRAY  0A + , C;
 13
 14
 15                        BASE !
```

```
Screen: 165
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 166
  0 ( Sound:  SOUND  SO.  FILTER!   )
  1
  2 BASE @ HEX
  3 0 VARIABLE AUDCTL
  4
  5 : SOUND  ( ch# freq dist vol --)
  6   3 DUP D20F C! 232 C!
  7   SWAP 10 * + ROT 2*
  8   D200 + ROT OVER C! 1+ C!
  9   AUDCTL C@ D208 C! ;
 10
 11 : SO. SOUND ;
 12
 13 : FILTER!                ( b -- )
 14   DUP D208 C! AUDCTL ! ;
 15                                ==>
```

```
Screen: 167
  0 ( Sound:  XSND  XSND4           )
  1
  2
  3 : XSND                ( voice# -- )
  4   2* D201 +
  5   0 SWAP C! ;
  6
  7
  8 : XSND4                       ( -- )
  9   D200 8 0 FILL
 10   0 FILTER! ;
 11
 12
 13 '( POS. )( : POS. 54 C! 55 ! ; )
 14
 15 BASE !
```

```
Screen: 168
  0 ( Utils:  STICK                    )
  1 BASE @ HEX
  2 LABEL STKARY
  3   0 , -1 , 1 , 0 ,
  4
  5 : STICK                ( n -- n n )
  6   278 + C@ 0F XOR
  7   DUP 2/ 2/ 3 AND
  8   2* STKARY + @
  9   SWAP 3 AND
 10   2* STKARY + @ ;
 11
 12 CODE STRIG                ( n -- f )
 13   B4 C, 00 C,  B9 C, 284 ,
 14   49 C, 01 C, 4C C, PUT0A , C;
 15 BASE !
```

```
Screen: 169
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 170
  0 CONTENTS OF THIS DISK:
  1
  2 PLAYER/MISSILES:          30 LOAD
  3 AUDIO EDITOR:             60 LOAD
  4 CHARACTER EDITOR:         90 LOAD
  5 CHARACTER SET WORDS:     120 LOAD
  6
  7 STANDARD CHARACTER SET   130 LIST
  8 SPACE SHIP IMAGES        132 LIST
  9
 10 PM EX. #1   ( BOUNCE )   140 LOAD
 11 PM EX. #2   (  SHIP  )   150 LOAD
 12
 13 ARRAYS ( FOR ALL    )    160 LOAD
 14 SOUNDS ( FOR AUDED )     166 LOAD
 15 STICK                    168 LOAD
```

```
Screen: 171
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 172
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 173
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 174
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 175
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 176
 0 ( Error messages                                              )
 1
 2 Stack empty
 3
 4 Dictionary full
 5
 6 Wrong addressing mode
 7
 8 Is not unique
 9
10 Value error
11
12 Disk address error
13
14 Stack full
15
```

```
Screen: 177
 0 Disk Error!
 1
 2 Dictionary too big
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 178
 0 ( Error messages                                              )
 1
 2 Use only in Definitions
 3
 4 Execution only
 5
 6 Conditionals not paired
 7
 8 Definition not finished
 9
10 In protected dictionary
11
12 Use only when loading
13
14 Off current screen
15
```

```
Screen: 179
 0 Declare VOCABULARY
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

HANDY REFERENCE CARD

valFORTH™ SOFTWARE SYSTEM

PLAYER-MISSILE GRAPHICS, CHARACTER EDITOR, & SOUND EDITOR

Player/Missile Command Summary

Note: Players and missiles are numbered 0 thru 3. The fifth player is numbered 4.

(PMINIT)	(addr res ---)	Initializes the player missile routines with PM memory specified by "addr" with "res" resolution.
PMINIT	(res ---)	Initializes the player missile routines with "res" resolution and with PM memory located at the first available memory below the display list.
PMBAS	(--- addr)	A variable pointing to player/missile memory which is set by (PMINIT) or PMINIT. It can be read from but not written to.
PLAYERS	(ON/OFF ---)	This command enables or disables the player/missile graphic display.
5THPLY	(ON/OFF ---)	This command turns (the fifth player mode) ON or OFF. If OFF, missiles take the colors of their corresponding players. If ON, all missiles take on the common color of playfield 3. The fifth player is numbered as four (4).
PLYCLR	(pl# ---)	Erases the specified player (0-3,4).
MSLCLR	(ml# ---)	Erases the specified missile (0-3).
PMCLR	(---)	Erases all players and all missiles.
MCPLY	(ON/OFF ---)	This command turns (the multiple color player mode) ON or OFF. See documentation for explanation.
PRIOR	(n ---)	Sets the priority of players and playfields. See documentation for legal settings.
PLYWID	(width pl# ---)	Sets the width of the specified player. Legal widths are normal (0 or 2), double (1), or quadruple (3).
MSLWID	(width ml# ---)	Sets the width of the specified missile. Legal widths are normal (0 or 2), double (1), or quadruple (3).
PMCOL	(pl# hue lum ---)	Sets the specified player to the color defined by "hue" and "lum".
BLDPLY	(addr len horz vert pl# ---)	Creates a player whose image is at "addr" with a length "len". The player is originally placed at the specified horizontal and vertical coordinates.
BLDMSL	(addr len horz vert ml# ---)	Creates a missile whose image is at "addr" with a length "len". The player is originally placed at the specified horizontal and vertical coordinates.
PLYLOC	(pl# --- horz vert)	Returns the horizontal and vertical coordinates of the specified player.
MSLLOC	(ml# --- horz vert)	Returns the horizontal and vertical coordinates of the specified missile.
PLYMV	(horz vert pl# ---)	Moves the specified player according to the horizontal and vertical offsets specified. A positive horizontal offset moves the player right, a negative one moves it left. Likewise, a positive vertical offset moves the player down and a negative one moves it up.
MSLMV	(horz vert ml# ---)	Moves the specified missile according to the horizontal and vertical offsets specified. See PLYMV above.
PLYPUT	(x y pl# ---)	Positions the specified player and location (x,y) on the video display.
PLYCHG	(addr len pl# ---)	This changes the image of the specified player to the image of length "len" at "addr".
PLYSEL	(addr # pl# ---)	This changes the image of the specified player to image number "#" in a table of images starting at address "addr".
PLYBND	(l r t b pl# ---)	Specified the left, right, top, and bottom boundaries of the specified player.
MSLBND	(l r t b ml# ---)	Specified the left, right, top, and bottom boundaries of the specified missile.
?BND	(--- n)	Returns the boundary status of the last player or missile moved. See documentation for a description of this value.
?PLYSTT	(pl# --- n)	Returns the boundary status of the last move of the specified player. See documentation for a description of this value.
?MSLSTT	(ml# --- n)	Returns the boundary status of the last move of the specified missile. See documentation for a description of this value.
?COL	(--- f)	Returns true (1) if any collisions have occurred since the last HITCLR command was issued.
?MXPF	(ml# --- n)	Returns 0 if the specified missile has not hit any playfields since the last HITCLR command. If any collisions have occurred, a status value is returned. See documentation.
?PXPF	(pl# --- n)	Returns 0 if the specified player has not hit any playfields since the last HITCLR command. If any collisions have occurred, a status value is returned. See documentation.
?MXPL	(ml# --- n)	Returns 0 if the specified missile has not hit any players since the last HITCLR command. If any collisions have occurred, a status value is returned. See documentation.
?PXPL	(pl# --- n)	Returns 0 if the specified player has not hit any other players since the last HITCLR command. If any collisions have occurred, a status value is returned.
HITCLR	(---)	Clears the collision registers to a no-collision state.

Audio Editor Command Summary

AUDED	(--)	Calls up the audio-palette program.

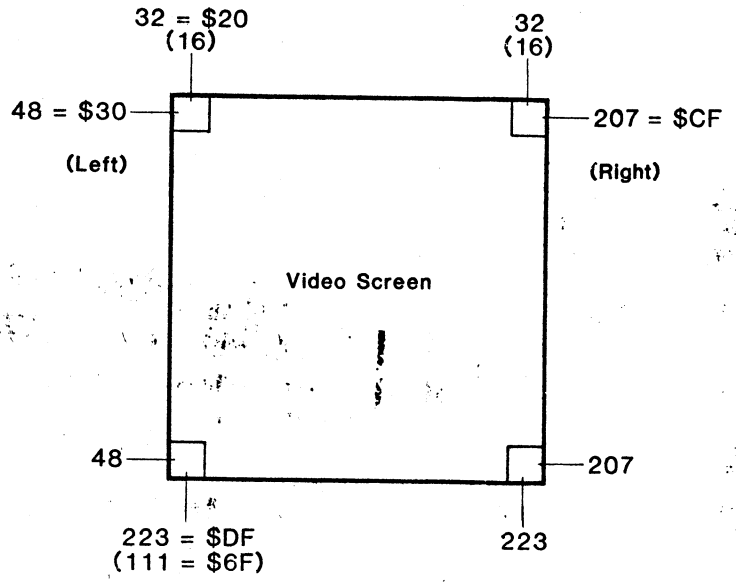
Character Editor Command Summary

CHAR-EDIT	(--)	Calls up the character editor.

PLAYER-MISSILE BOUNDARY MAP

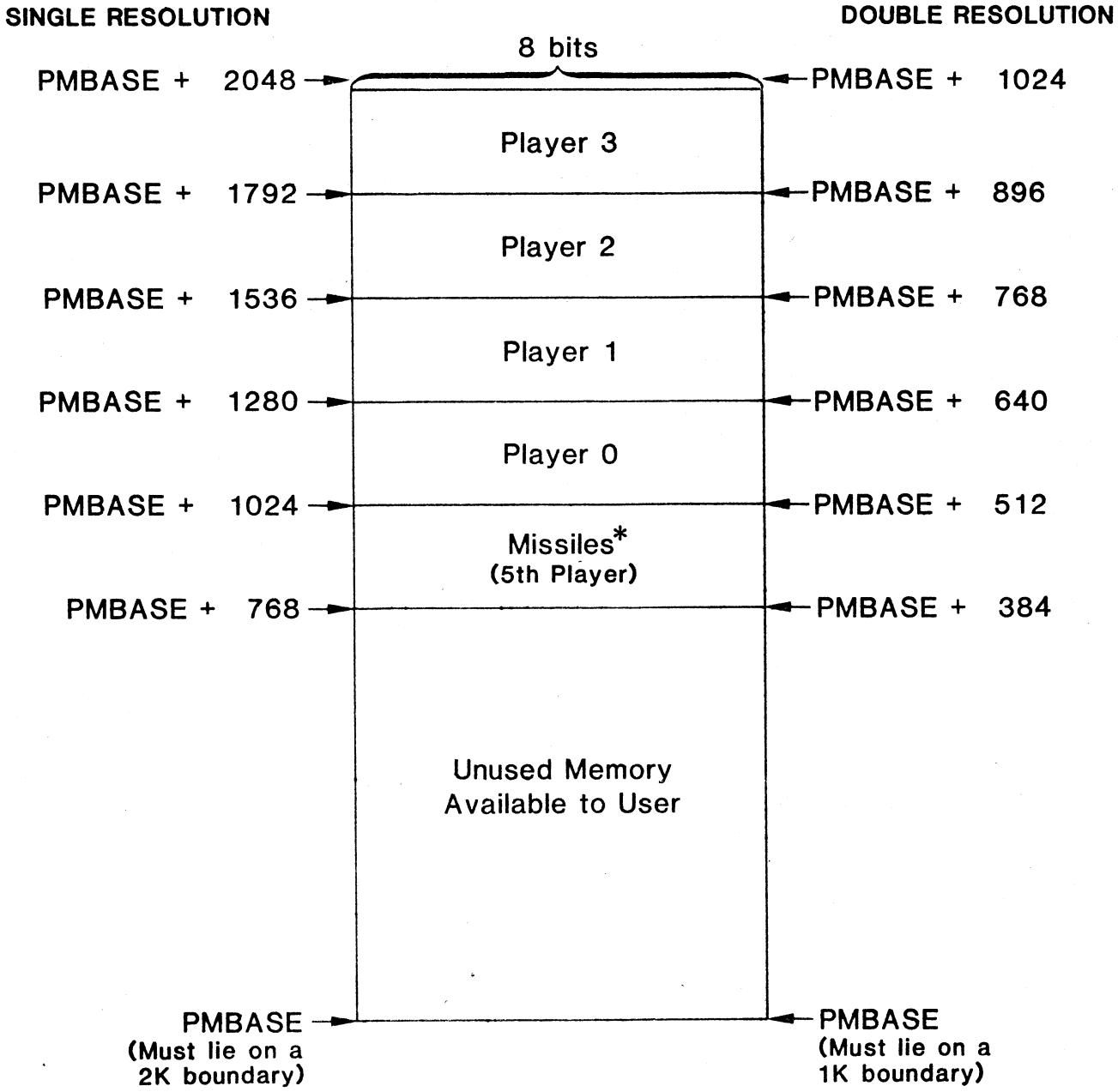
valFORTH

(Double resolution values are in parentheses)



PLAYER-MISSILE Memory Map

valFORTH™



*Note: All missiles occupy the same memory location. This is possible because unlike players which are 8 bits wide and fill an entire byte, missiles are only two bits wide. Four missiles can therefore be represented in the same amount of memory as a single player.

Byte form: ! m3 ! m2 ! m1 ! m0 !